Software Schedules: Nailing Jello to the Wall

Marghi Hopkins

NASA Project Management Challenge 2009

© 2009 The Aerospace Corporation

Why Are We Having This Conversation?

Software schedules - still risky after all these years

- Notwithstanding years of
  - *Lessons learned*
  - *Process improvement*
  - *Metrics*
- We still routinely face
  - *Large uncertainties in schedule estimation*
  - *Problematic schedule tracking*
- Is there any help?

Presentation Topics

Related to software and schedules

- Effects and trends in software scheduling
- The system development lifecycle
- The software subsystem
- Symptoms and causes of scheduling problems
- Several approaches to developing and maintaining accurate software schedules

Twenty Years And 40+ Missions: Summary Of Cost/Schedule Study Results*

Effects of scheduling overruns

- 10% schedule growth corresponds to 12% cost growth
- Assumptions and allotments for schedule growth
  - *General rule of thumb is 1 month per year*
  - *NASA/JPL guidance is 1.8 months per year*
  - *Four of six telescope missions exceeded NASA/JPL guidance*
  - *Five of six telescope missions exceeded general rule of thumb*
- Reliability of estimates improves over the life cycle but still contains large uncertainty
- Greatest growth occurs in Phase D during Integration and Test
- Short schedules and increased complexity invite failure
  - *Failed and impaired missions had short schedules or a combination of short schedules and high complexity*

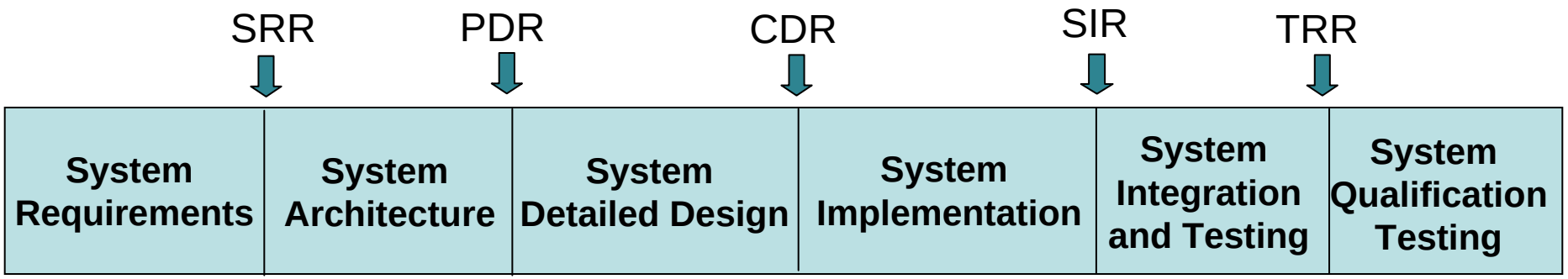
*Bearden, Perspectives on NASA Mission Cost and Schedule Performance Trends, Presentation at GSFC Symposium, 3 June 2008 [http://ses.gsfc.nasa.gov/SE_Seminar_2008.htm]

Trends: More Complex, More Complicated

- Complex
  - *1 : composed of two or more parts*
  - *2 : difficult to analyze, understand, or explain*
- Complicated
  - *1 : consisting of parts intricately combined*
  - *2: hard to separate, analyze, or solve*
- Classical Systems Engineering assumes a closed system: decomposable, linear, predictable
- Newer problems tend to reside in open systems that exhibit predictable patterns of behavior, unpredictable patterns of behavior, or a combination of both

The **Big Picture**

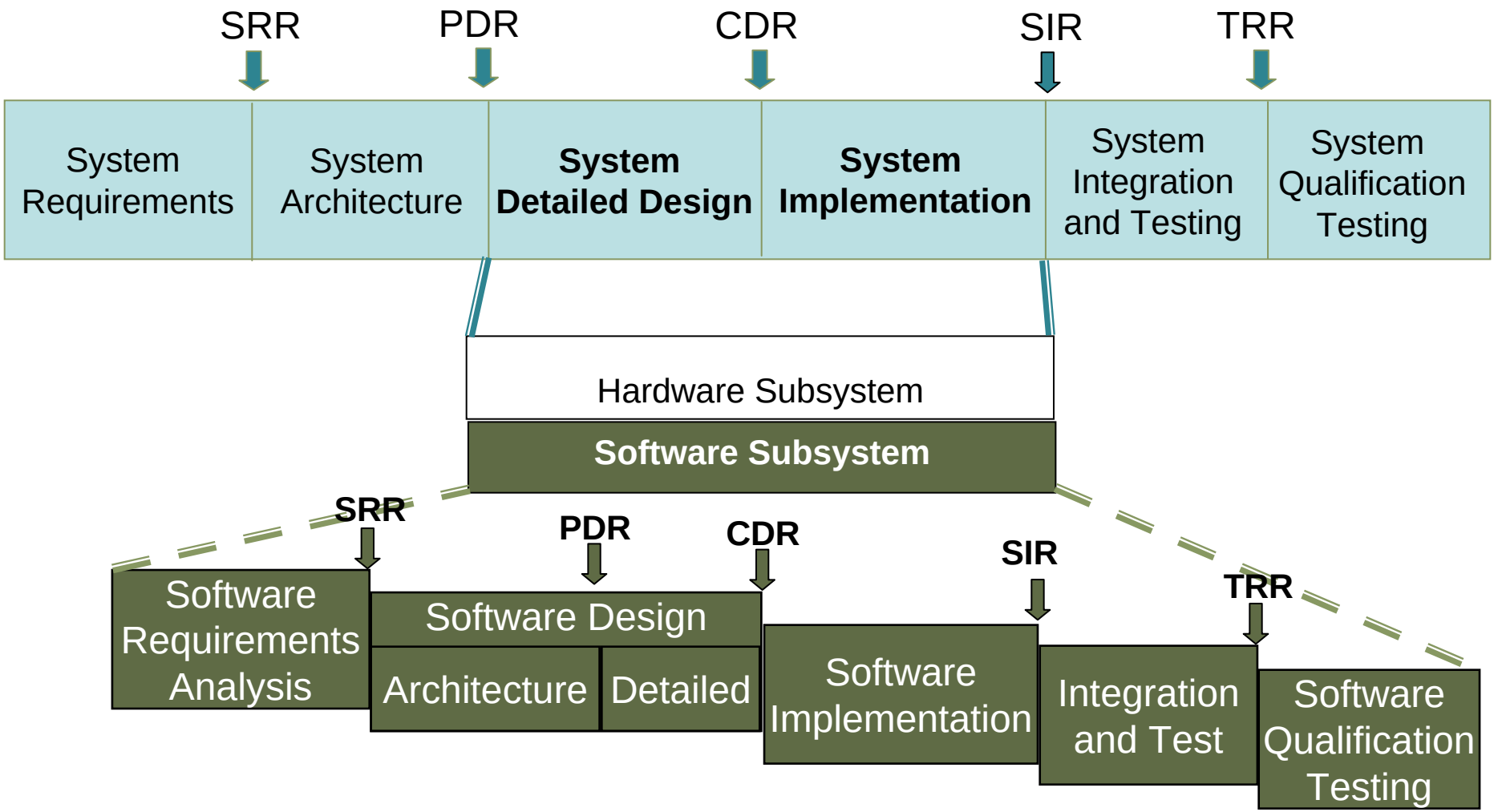
The system development lifecycle



SRR – System Requirements Review
PDR – Preliminary Design Review
CDR – Critical Design Review
SIR – System Integration Review
TRR – Test Readiness Review

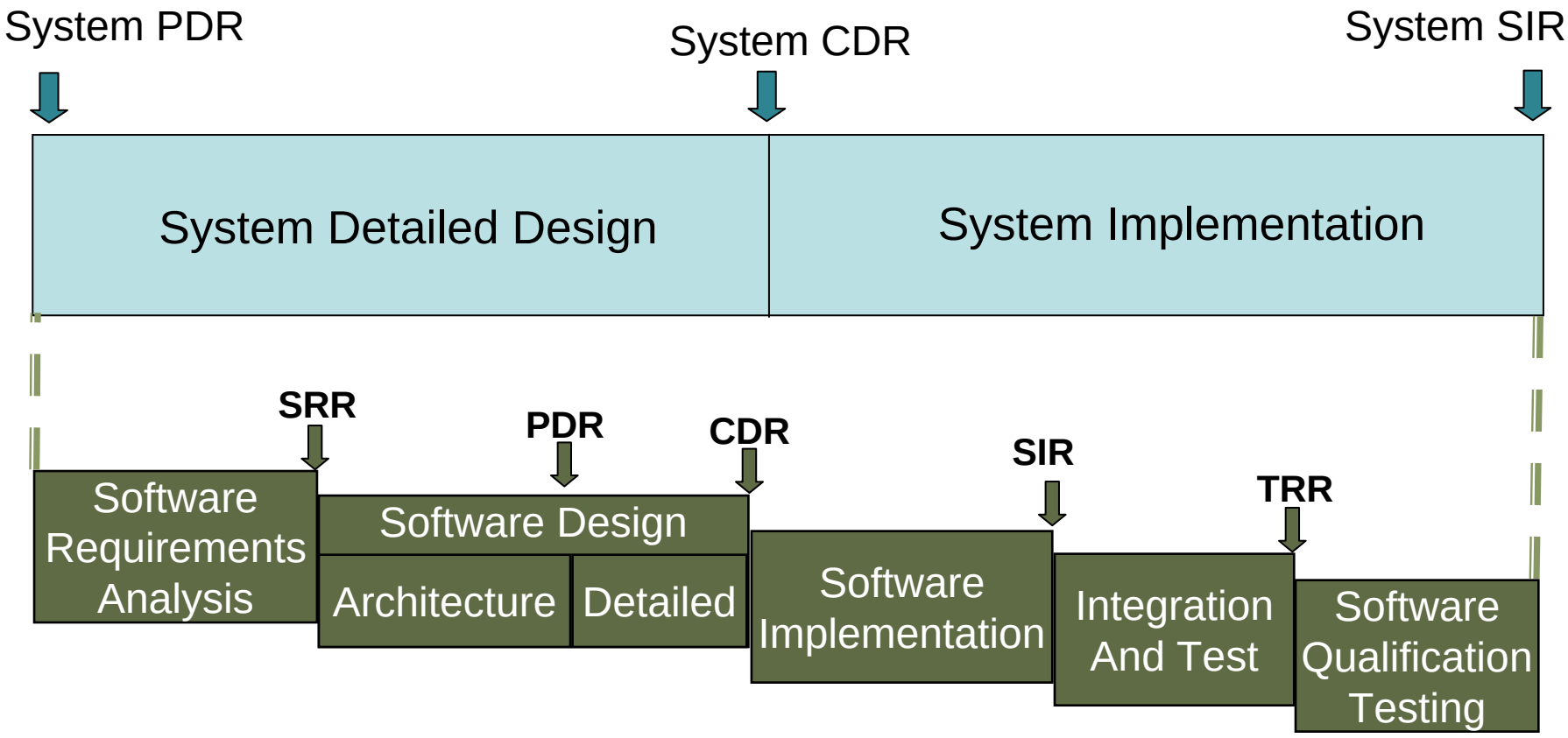
The Section On Which We Are Focused

The software subsystem



Looking Closer

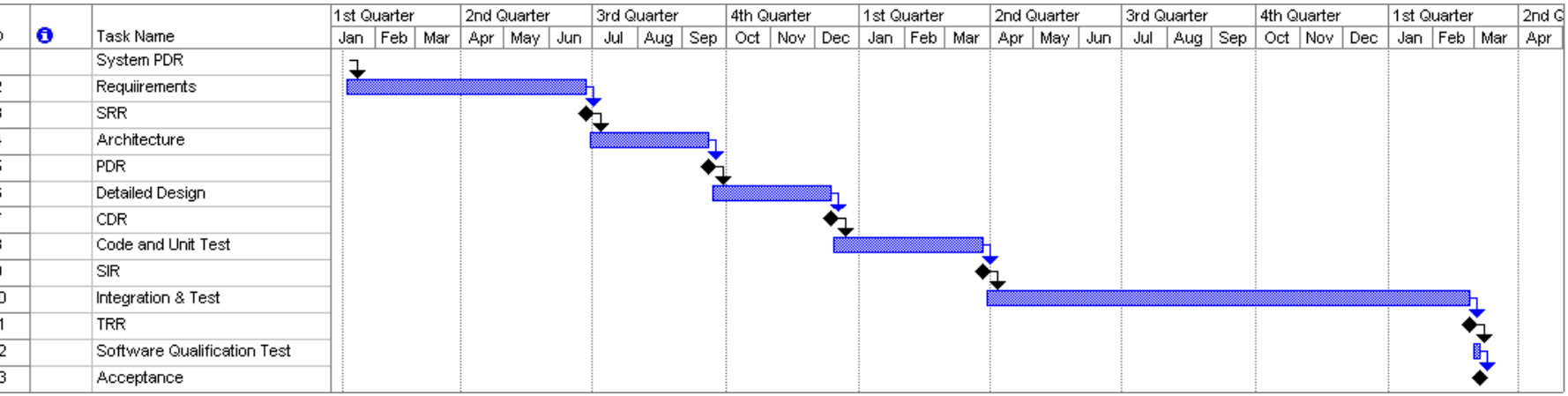
The software subsystem

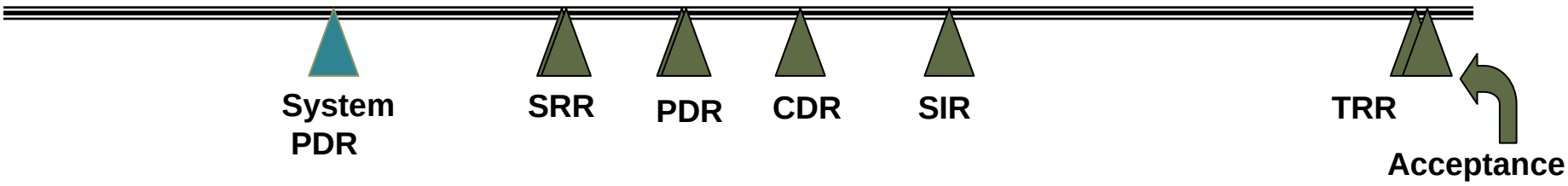


Establishing A Schedule And Milestones

The software subsystem

ID	Task Name
1	System PDR
2	Requiirements
3	SRR
4	Architecture
5	PDR
6	Detailed Design
7	CDR
8	Code and Unit Test
9	SIR
10	Integration & Test
11	TRR
12	Software Qualification Test
13	Acceptance

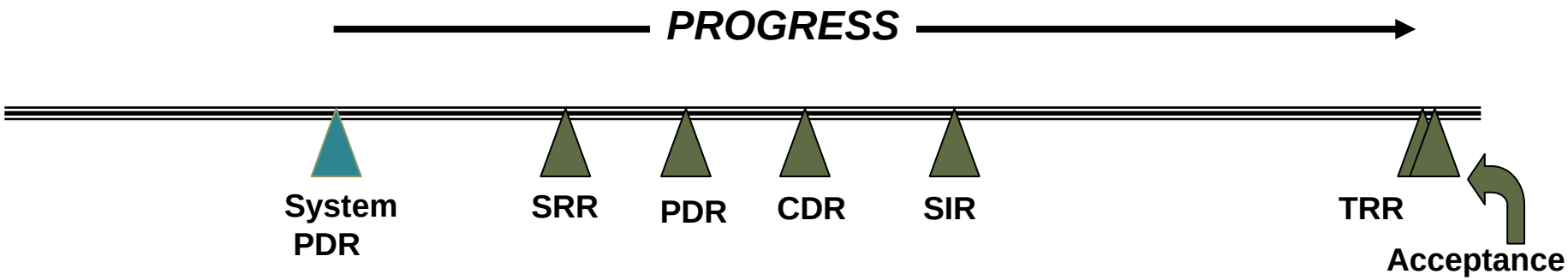




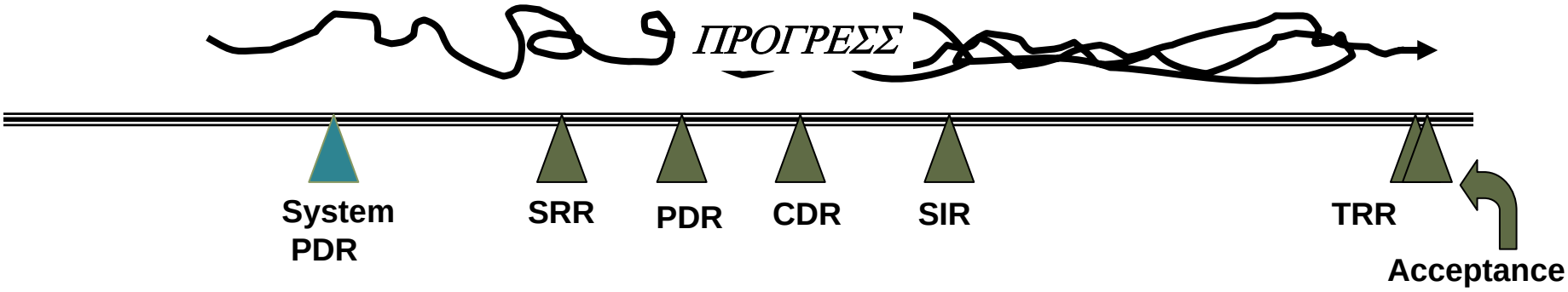
Tracking The Schedule

The software subsystem

In a perfect world:



In our world:



Typical Software Scheduling And Tracking Problems

- The software schedule seems reasonable but keeps changing
  - *The project keeps receiving new requirements*
  - *Coding is finished, but integration and test is taking more time than expected*
- The software schedule is not reasonable and keeps slipping
  - *COTS and reused code didn't reduce the schedule as much as predicted*
- Tracking the schedule is difficult
  - *The code is 90% done*
  - *The software status hasn't changed since the last reporting period*

Software Scheduling And Tracking Problems

Well-known causes

- Poor estimates,
  - *Ill-defined requirements*
  - *Changing requirements*
  - *Failure to understand the problem*
  - *Inexperienced team*
  - *Lack of metrics*
  - *Poor or inappropriate use of metrics*
  - *Lack of review by technical people*
- Poor tracking
  - *No tracking is done*
  - *Inaccurate status of tasks*
  - *Overly optimistic*
  - *Improper use of Earned Value Measurement*

Software Scheduling And Tracking Problems

Less well-recognized causes

- The software schedule is not properly synchronized with the system schedule
  - *A software schedule cannot realistically begin until after the System Preliminary Design Review (PDR), when the Review Board approves allocation of functionality to hardware and software systems*
  - *Starting the software schedule before System PDR generally results in slips, rework, and rescheduling*
- Failure to include the requirements-design-implement-test cycle *at even the lowest levels*, including bug fixes and requested changes

Software Scheduling And Tracking Problems

Personal Experiences

- Inadequate or no time in the schedule for the requirements-design-implement-test cycle, especially during integration and test
  - *Little or no unit testing was done*
  - *Bugs were not correctly fixed or new bugs were installed because requirements and design were not reviewed*
- Forward progress reverses when problems must be fixed, but tracking does not include options for accurately conveying project status
  - *No consideration of "are we going in the right direction?"*
  - *When 100 lines of code are counted as complete, does anybody ever subtract that 100 lines when the module fails in integration? No.*

Using Technology To Enhance Project Scheduling

Several Examples

- Evidence Based Scheduling* to get a confidence distribution curve showing you will meet the target completion date
  - *Maximum 16 hour tasks – this forces you to do the detailed design*
  - *Capture individual project member's elapsed time for each task*
  - *Calculate estimate/actual ratios to determine how fast the task was done relative to the estimate; perform a Monte Carlo simulation for each project member using data from the past 6 months*
  - *Calculate each project member's probability of completing assigned tasks on a given date; the developer who finishes last determines when the team is done*
- Using this approach produces a realistic project schedule

***Spolsky, Evidence Based Scheduling, Article on the Joel on Software Homepage, 26 October 2007 [http://www.joelonsoftware.com]**

Benefits Of Evidence Based Scheduling

- Significantly enhanced accuracy
- Empirical evidence that allows you to understand project status in concrete terms
- Real-time performance data that forces you to prioritize and help control scope creep
  - *Given a bunch of blocks and a box that won't hold them, you must get a bigger box or discard some of the blocks.*
- Realistic schedules that permit active project management to meet milestones, not just post-mortem analysis
- Once procedures are set up, they can be quickly and easily deployed in later phases or other projects

Non-Technological Enhancements To Project Scheduling

Several Examples

- Track progress using something other than, or in addition to, lines of code
  - *Number of requirements met*
  - *Number of Function Points implemented*
  - *Number of test cases successfully executed*
  - ***Number of Use Cases that can be executed in a build***
    - Not difficult to calculate
    - Use Cases are almost universally used now for requirements analysis and design, so you use existing information
    - Helps enforce using the requirements-design-implementation-test cycle at all levels: system, CSCI, CSC, and unit
- Functional, rather than quantitative status reports provide a more reliable view of progress than the sum of number of lines of code generated

In Summary

- The software schedule should begin after system PDR, not before
- Provide sufficient time for the requirements-design-implement-test cycle at all levels

- Project Management should include metrics that provide a realistic view of project status
  - *Functional milestones, not amount of code*
  - *Current loads on all team members—and disparities between them*
  - *Specific identification of open issues and cost overruns*
  - *Dynamic timelines that show the impact of a slipped schedule*